



Clase 5. Python

Controladores de Flujo 2

***RECUERDA PONER A GRABAR LA
CLASE***





OBJETIVOS DE LA CLASE

- Identificar el proceso de iteración en programación
- Diferenciar sentencia while de sentencias while-else
- Implementar instrucción break
- Diferenciar entre instrucción continue y pass
- Implementar sentencia for, range y for-else-break-continue-pass

CRONOGRAMA DEL CURSO

Clase 4



Controladores de flujo 1



MAYORÍA DE EDAD



MARVEL VS CAPCOM

Clase 5



Controladores de flujo 2



NÚMEROS



EJERCICIOS



CONTROL DE FLUJO

Clase 6



Conjuntos y diccionarios



SETS



DICTS

SENTENCIAS ITERATIVAS

Repetir

Las computadoras se usan a menudo para automatizar tareas repetitivas.

Realizar repetidamente tareas idénticas o similares sin cometer errores es algo que las computadoras hacen bien y que los seres humanos hacemos limitadamente.



Iterar



En matemática, se refiere al proceso de iteración de una función, es decir, aplicando la función repetidamente, usando la salida de una iteración como la entrada a la siguiente.

Iterar

En programación, Iteración es la **repetición** de un **segmento de código** dentro de un programa de computadora. Puede usarse tanto como un término genérico (como sinónimo de repetición) como para describir una forma específica de repetición.



```
31
32 self.file = None
33 self.fingerprints = set()
34 self.logdupes = True
35 self.debug = debug
36 self.logger = logging.getLogger(__name__)
37 if path:
38     self.file = open(os.path.join(path, 'requests.txt'),
39                     'a')
40     self.file.seek(0)
41     self.fingerprints.update(os.listdir(path))
42
43 @classmethod
44 def from_settings(cls, settings):
45     debug = settings.getbool('DEBUG', False)
46     return cls(job_dir(settings), debug)
47
48 def request_seen(self, request):
49     fp = self.request_fingerprint(request)
50     if fp in self.fingerprints:
51         return True
52     self.fingerprints.add(fp)
53     if self.file:
54         self.file.write(fp + os.linesep)
55
56 def request_fingerprint(self, request):
57     return request_fingerprint(request)
```




Tenemos una **base de datos enorme** y queremos encontrar **un dato** en especial para consultar.

Como no existe una forma mágica para encontrar el dato directamente, el programa deberá recorrer los datos uno a uno y compararlos hasta dar con el que buscamos iterando o repitiendo el mismo proceso desde el inicio comparando a ver si es el que queremos o no, así hasta que encuentre el que queremos.



Existen algoritmos que permiten ahorrarnos tiempo e iteraciones, pero en esencia sigue recorriendo uno a uno, la diferencia es que nosotros tardaríamos muchas horas, y el programa unos segundos.

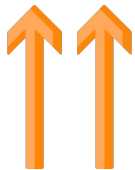
¡Así que vamos a aprovecharnos de esto!



- ***WHILE***
- ***FOR***

WHILE (no sabes las la cantidad)

Ejemplo

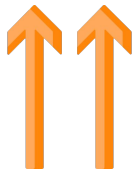


Sentencia While

Vamos a comenzar con la sentencia iterativa más básica **While** (mientras).

Se basa en repetir un bloque de código a partir de evaluar una **condición lógica**, siempre que ésta sea **True** (al igual que la sentencia **if**).

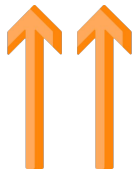
Nosotros como programadores debemos decidir el momento en que la condición cambie a **False** para hacer que el While finalice su ejecución y así salir de la iteración, de lo contrario estaríamos frente a un **bucle infinito**.



Flujo de ejecución

Más formalmente, el flujo de ejecución de una sentencia **while** es el siguiente:

1. Evalúa la condición, devolviendo **False** o **True**.
2. Si la condición es **False**, se sale de la sentencia **While** y continúa la ejecución con la siguiente sentencia.
3. Si la condición es **True**, ejecuta cada una de las sentencias en el bloque de código y regresa al paso 1.

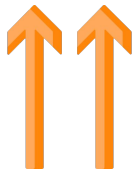


Bucle

Este tipo de flujo se llama **bucle** porque el **tercer paso del bucle vuelve arriba**.



Si la condición es falsa la primera vez que se pasa el bucle, las sentencias del interior del bucle no se ejecutan nunca.



Ejemplo



Analicemos qué pasó

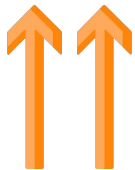
```
>>> num = 5
>>> while num > 0:
    print(f"{num}")
    n -= 1

>>> print("Terminó el conteo!")
```

1. Declaramos una variable num y le asignamos el valor int 5.
2. Usamos la sentencia while para indicar que mientras que num sea mayor a 0 entremos al bloque de código.
3. Al evaluar num contra 0 nos indica que es True



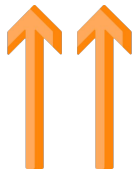
CODER HOUSE



¿Qué pasó?



4. Ingresamos al bloque de código, imprimimos num y le restamos 1 a num
5. Volvemos a repetir desde el paso 2 hasta que num deje de ser mayor a 0
6. Cuando la operación relacional de False saldremos del bucle
7. Imprimimos por pantalla Terminó el conteo!
8. Termina nuestro programa



Más ejemplos con While



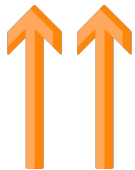
```
>>> n = 0
>>> while n <= 5:
    n += 1
    print("N vale ", n)
```

¿Y un bucle infinito?

```
>>> while True:
    print("Esto es un
    bucle infinito!!!!!!")
```

Para escapar un bucle infinito generalmente se usa **ctrl + c**

WHILE- ELSE



Sentencia While-else

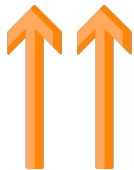


```
>>> while condicion:
    # instrucciones de while
>>> else:
    # instrucciones de while-else
>>> si no se abortó con break:
    # Instrucciones del while-else
```

Algo interesante que tiene python es que podemos encadenar un **else** (si no) al final de un bucle **while**.

Este **else** sirve para ejecutar un bloque de código cuando el bucle while tenga una condición **False** o haya **terminado** y no haya sido forzado a salir mediante un **break**.

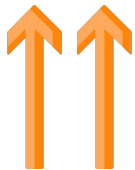
CODER HOUSE



¿Qué pasó?

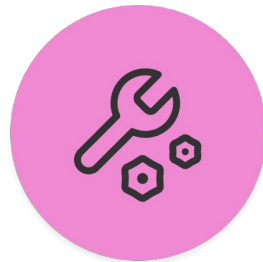


```
>>> chance = 1
>>> while chance <= 3:
    txt = input("Escribe SI: ")
    if txt == "SI":
        print("Ok, lo conseguiste en el intento", chance)
        break
    chance += 1
else:
    print("Has agotado tus tres intentos")
```



Sentencia While-else

1. Declaramos una variable **chance** y le asignamos el valor **int 1**.
2. Usamos la sentencia **while** para indicar que **mientras** que **chance** sea **menor** a 3 entremos al bloque de código.
3. Le pedimos al usuario que ingrese una palabra con input
4. Si la palabra es **“SI”** ingresa al condicional **if**
5. Si ingresa, imprime que lo consiguió en el intento tal y rompe el bucle con **break**
6. Si la condicional es **False** vamos a sumar uno a las chances y repetir desde el paso 2
7. Sí **chance** es mayor a 3, entramos en el **else** e imprimimos



¡Números!

Mostrar la suma de los números ingresados

Tiempo estimado: 10 minutos



¡Números!

Tiempo estimado: 10 minutos

Escribir un programa que le pregunte al usuario números hasta que ingrese el 0, cuando lo haga mostrar por pantalla la suma de todos **los números ingresados**.

Nota: Para preguntarle al usuario, recuerda usar input

INSTRUCCIONES

- ***break***
- ***continue***
- ***pass***



Bucles en Python

Usar bucles en Python nos permite automatizar y repetir tareas de manera eficiente.

Sin embargo, a veces, es posible que un **factor externo influya en la forma en que se ejecuta su programa.**



Instrucciones

Cuando esto sucede, es posible que prefiramos que nuestro programa cierre un bucle por completo, omita parte de un bucle antes de continuar o ignore ese factor externo.

Para hacer estas acciones python nos brinda las **instrucciones break, continue y pass.**

BREAK



Break

Comencemos por uno de los más sencillos y más utilizados: El **break**.

En Python, la instrucción **break** le proporciona la oportunidad de **cerrar un bucle** cuando se **activa** una **condición externa**. Debe poner la instrucción **break** dentro del bloque de código bajo la instrucción de su bucle, generalmente **después** de una sentencia **if** condicional.



Ejemplo



```
>>> n = 5:
>>> while n < 10:
    n -= 1
    if n == 2:
        print("ahora que n vale 2 salimos")
        break
    print("n vale ", n)
```

Analicemos qué pasó



¿Qué pasó?



1. Declaramos una variable **n** y le asignamos el valor **int** 5.
2. Usamos la sentencia **while** para indicar que **mientras** que **n** sea **menor** a 10 entremos al bloque de código.
3. Le restamos 1 a n
4. Usamos una sentencia **if** condicional para igualar n a 2
5. Si da **True** imprime ahora que n vale 2 salimos y con **break** sale del bucle, ya no se ejecuta el resto
6. Si da **False** imprime que n vale n y vuelve al paso 2.



SENTENCIA BREAK

Es hora de programar mentalmente.

TIEMPO: 10 MIN

SENTENCIA BREAK

Actividades
colaborativas



Tiempo: 10 minutos

Consigna: Explica qué sucedió en este caso:

```
>>> c = -3:
>>> while True:
    c += 1
    if c == 2:
        print("ahora que c vale 2 salimos")
        break
    print("c vale ", c)
```

CONTINUE



Continue

La instrucción **continue** da la opción de **omitir** la parte de un bucle en la que se **activa** una condición externa, pero continuar para completar el resto del bucle. Es decir, la **iteración actual del bucle se interrumpirá, pero el programa volverá a la parte superior del bucle.**

Debe poner la instrucción **continue** dentro del bloque de código bajo la instrucción de su bucle, generalmente **después** de una sentencia **if** condicional.



Continue



Veamos un ejemplo:

```
>>> n = 0:
```

```
>>> while n < 10:
```

```
    n += 1
```

```
    if n == 2:
```

```
        print("Continuamos con la siguiente iteración")
```

```
        continue
```

```
    print("n vale ", n)
```

Analizamos qué pasó



¿Qué pasó?

1. Declaramos una variable **n** y le asignamos el valor **int 0**.
2. Usamos la sentencia **while** para indicar que **mientras** que **n** sea **menor** a 10 entremos al bloque de código.
3. Le sumamos 1 a n
4. Usamos una sentencia **if** condicional para igualar n a 2
5. Si da **True** imprime **Continuamos con la siguiente iteración** y con **continue** rompe la iteración, pero no de todo el bucle, sólo de esta iteración
6. Si da **False** imprime que n vale n y vuelve al paso 2.

PASS



Pass

Cuando se activa una condición externa, la instrucción **pass** permite manejar la condición **sin que el bucle se vea afectado** de ninguna manera; todo el código continuará leyéndose a menos que se produzca la instrucción break u otra instrucción.

Debe poner la instrucción **pass** dentro del bloque de código bajo la instrucción de su bucle, generalmente **después** de una sentencia **if** condicional.



Ejemplo

```
>>> n = 0:  
>>> while n < 10:  
    n += 1  
    if n == 2:  
        pass  
    print("n vale ", n)
```

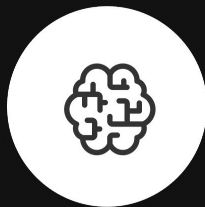


¿Qué pasó?



1. Declaramos una variable **n** y le asignamos el valor **int 0**.
2. Usamos la sentencia **while** para indicar que **mientras** que **n** sea **menor** a 10 entremos al bloque de código.
3. Le sumamos 1 a n
4. Usamos una sentencia **if** condicional para igual n a 2
5. Si da **True** con la instrucción pass le indica al programa que continúe ejecutando el bucle e **ignore** el hecho de que la variable n se evalúa como equivalente a 2 durante una de sus iteraciones
6. Si da **False** imprime que n vale n, le resta 1 y vuelve al paso 2.

CODER HOUSE



PARA PENSAR

¿Qué pasó en este ejemplo?

```
>>> c = -3
>>> while c < 10:
    c += 1
    if c == 2:
        pass
    print("c vale ", c)
```

FOR
(repetir, pero conociendo la cantidad de repeticiones)



Sentencia For

Ahora seguiremos con la sentencia iterativa que podríamos decir es la más usada **For** (para).

Se utiliza para recorrer los elementos de un objeto iterable (lista, tupla...) y ejecutar un bloque de código, o sea, tiene un número predeterminado de veces que itera.



Sentencia For

En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.

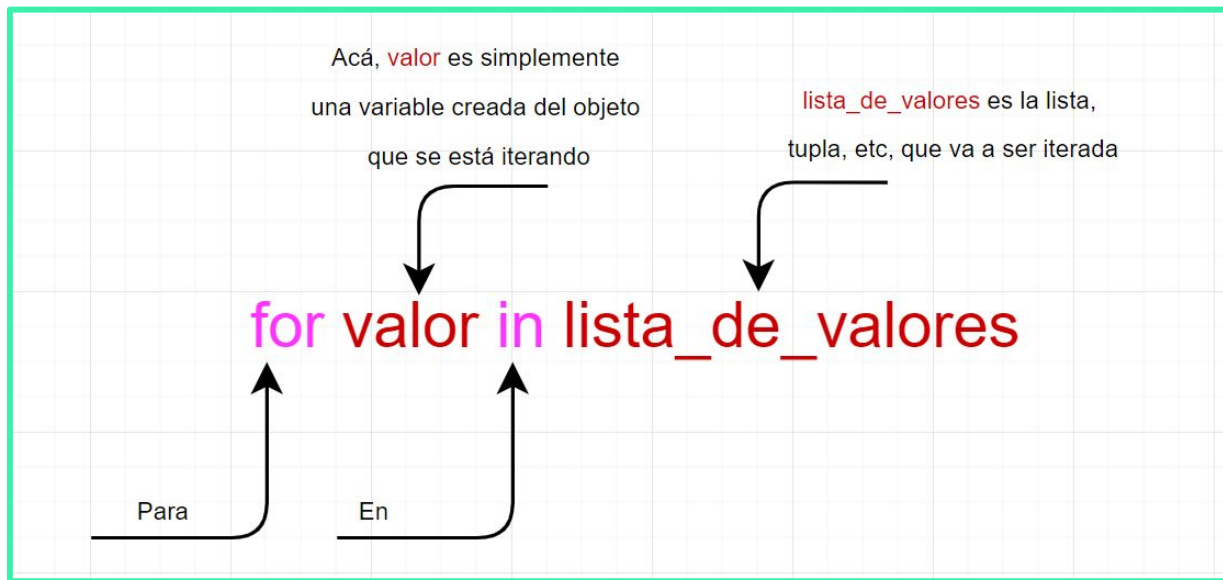
```
>>> lista = [1,2,3,4,5]
```

```
>>> for valor in lista:
```

```
    print("Soy un item de la lista y valgo ", valor)
```



Ejemplo gráfico



valor simplemente es una copia local, no afecta fuera del bucle a menos que se devuelva el valor



Ejemplo

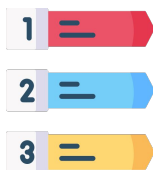
```
>>> lista = [0,1,2,3,4,5,6,7,8,9,10]
>>> for num in lista:
    print("Soy un valor de la lista y valgo ", num)
    num *= 5
    print("Soy un valor de la lista y ahora valgo ", num)
```



Modificando la lista

```
>>> indice = 0
>>> numeros = [0,1,2,3,4,5,6,7,8,9,10]
>>> for numero in numeros:
    numeros[indice] *= 5
    indice += 1
    # tambien se puede con indice = numero
>>> print(numeros)
```

ENUMERATE



Enumerate

La función incorporada `enumerate(lista/tupla_de_valores)` toma como argumento un objeto iterable y retorna otro cuyos elementos son tuplas de dos objetos:

1. El primero de los dos indica la posición de un elemento perteneciente a el objeto iterable, es decir, el índice.
2. El segundo, el elemento mismo.



Esto se conoce como lectura secuencial de clave y valor, lo vamos a usar bastante en el futuro!

CODER HOUSE

1 =

2 =

3 =

For + Enumerate



Si quisiéramos modificar la lista:

```
lista = [66,-3,35,96]
#---Pos ,   elem
for indice, numero in enumerate(lista):

    print(f"INDICE: {indice}")
    print(f"NUMERO: {numero}\n\n")

    print(f"-----> {lista[indice]}\n\n-----")
```



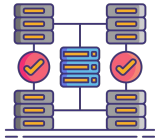
Sentencia For

Si quisiéramos recorrer un string:

```
>>> texto = "Hola Mundo, estoy usando for en Python"
>>> for letra in texto:
    print(letra)

>>> texto2 = ""
>>> for letra in texto:
    texto2 = letra * 2
>>> print(texto2)
```

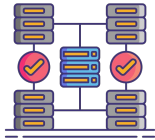
RANGE



¿Qué es?

Como vimos, el **for** en python **necesita** una colección de datos para poder utilizarlo, en otros lenguajes necesitamos solamente un número para indicar las iteraciones a cumplir.

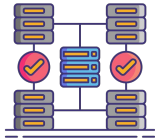
Para simular estos casos Python nos provee de una función denominada **range()** (rango) el cual **representa una colección de números inmutables.**



Constructores para crear objetos Range



1. range(**fin**): Crea una secuencia numérica que va desde 0 hasta fin - 1.
 - a. for numero in **range**(10)
2. range(inicio, fin): Crea una secuencia numérica que va desde inicio hasta fin - 1.
 - a. for numero in range(5, 10)
3. range(inicio, fin, [paso]): Crea una secuencia numérica que va desde inicio hasta fin - 1. Si además se indica el parámetro paso, la secuencia genera los números de paso en paso.
 - a. for numero in range(0, 20, 2)



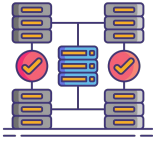
Ventajas



```
>>> range(0,10)
```

Parece ser exactamente una lista que va de 0 a 10, pero range interpreta el inicio y fin en tiempo de ejecución y eso le da ventaja contra la lista.

Si tuviéramos una lista de 0 a 10000 estaría ocupando muchísimo espacio en memoria.



Ventajas



```
>>> range(0,10000)
```

Ahora, si hiciéramos `range(0,10000)`, `range` interpreta esto en tiempo de ejecución, es decir cuando se ejecuta el 0 se crea el 0 cuando se ejecuta el 1 se crea el 1 y se elimina el 0 y así continuamente, y esto no ocupa memoria



For-else

Igual que en la sentencia while podemos usar un **else** al final de la iteración

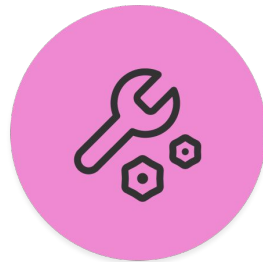
```
>>> for numero in range(10):  
    print("Numero vale ",numero)  
else:  
    print("Se terminó de iterar y numero vale: ", numero)
```



For-break-continue-pass

```
>>> for numero in range(10):  
    if numero == 2:  
        continue  
    elif numero == 8:  
        break  
    print("Numero vale ",numero)  
else:  
    print("Se terminó de iterar y numero vale: ", numero)
```

Igual que en la sentencia **while** podemos usar también las instrucciones **break continue** y **pass**



Ejercicios

Realizar los siguientes ejercicios

Tiempo estimado: 15 minutos

Ejercicios varios

Desafío
generico



Haremos el siguiente listado de ejercicios:

1. Escribir un programa que enumere los países de la siguiente lista:-----3
 - a. `paises = ['Canada', 'USA', 'Mexico', 'Australia', 'Argentina', 'China', 'India']`
2. Crear un bucle que sume los pares del 0 al 100-----1
3. Imprimir por pantalla los números del 1 al 10 al revés-----2
4. Pedirle a un usuario que ingrese un número, y devolver los dígitos totales del número
 - a. Por ejemplo, si el número es 75869, la salida debería ser 5.-----4

Nota:

Para imprimir por pantalla al revés se debe usar el mayor número, luego el menor, y el paso sería con `-1 range(mayor, menor, -1)`

CODER HOUSE

PUESTA EN COMÚN

¡Analicemos por última vez!

```
n = 0
while n < 10:
    if (n%2) == 0:
        print(n, 'es un número par')
    else:
        print(n, 'es un número impar')
    n += 1
```

¡Ahora podemos entender que hace este programa que vimos hace unas
clases!



¡CONTROL DE FLUJO!

Breve resumen de la consigna del desafío.

¡CONTROL DE FLUJO!

Formato: Documento que tiene el nombre de “Desafío entregable”, debe tener el nombre “Idea+Apellido”.

Sugerencia: Haz una copia del documento para trabajar.

Desafío
entregable



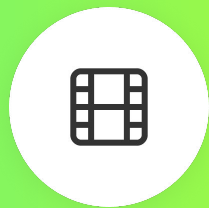
>> Consigna: Realizar los ejercicios del documento disponible en la carpeta.

 [Desafío entregable 3](#)

>>Aspectos a incluir en el entregable:Copia del documento con tus respuestas.

¿PREGUNTAS?





***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***



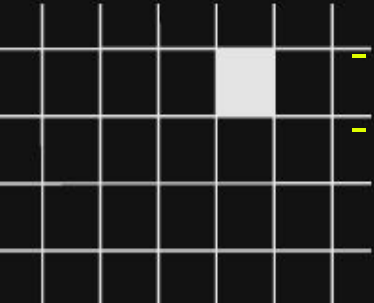
- Artículo: [Iteración](#)
- Artículo: [While - Break - Continue](#)
- Artículo: [For](#)
- Artículo: [Range](#)
- [EjemploClase](#)





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- 
- Iteración
 - While
 - Instrucciones break - continue - pass
 - For



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE